PAT-NO: JP408101777A
DOCUMENT-IDENTIFIER: JP 08101777 A
TITLE: INSTRUCTION STRING OPTIMIZING DEVICE
PUBN-DATE: April 16, 1996

INVENTOR-INFORMATION:

NAME	COUNTRY
IDE, YUKIHIRO	
YOSHIDA, TAKASHI	

ASSIGNEE-INFORMATION:

NAME	COUNTRY
TOSHIBA CORP	N/A

APPL-NO: JP06261394
APPL-DATE: September 30, 1994

INT-CL (IPC): G06F009/45 , G06F001/32

ABSTRACT:

PURPOSE: To perform the optimization processing for reduction of the power consumption in the stage of generation of a control program for information processor by providing an instruction string analysis means and an instruction change means.

CONSTITUTION: An intermediate code is generated, and next, it is discriminated whether each instruction has substitutive instructions registered in a library or not. When it is discriminated that the instruction has them, the library is retrieved to select the instructions which can be substituted for the original instruction. The power consumption of the original instruction and those of instructions retrieved from the library are tentatively calculated, and calculation results are compared with one another to select the instruction which minimizes the power consumption. It is discriminated whether optimization of all data is completed or not after the end of instruction substitution; and if it is completed, a program after the optimization processing is outputted. This control program is stored in a program memory 2620 of an information processor, and this program is used to control a CPU 2610, thus reducing the power consumption of an instruction bus 2632.

(19)日本国特許庁（ＪＰ）　(12)**公開特許公報（Ａ）**　(11)特許出願公開番号

特開平8－101777

(43)公開日　平成８年(1996)４月16日

(51)Int.Cl.[6]		識別記号	庁内整理番号	ＦＩ			技術表示箇所
Ｇ０６Ｆ	9/45						
	1/32						
			7737－5B	Ｇ０６Ｆ	9/44	３２２　Ｆ	
					1/00	３３２　Ｚ	

審査請求　未請求　請求項の数９　ＦＤ　（全 26 頁）

(21)出願番号　特願平6－261394

(22)出願日　平成６年(1994)９月30日

(71)出願人　000003078
株式会社東芝
神奈川県川崎市幸区堀川町72番地

(72)発明者　井　出　進　博
神奈川県川崎市幸区小向東芝町１　株式会社東芝研究開発センター内

(72)発明者　吉　田　華
神奈川県川崎市幸区小向東芝町１　株式会社東芝研究開発センター内

(74)代理人　弁理士　佐藤　一雄　（外３名）

(54)【発明の名称】　命令列最適化装置

(57)【要約】

【目的】　情報処理装置用制御プログラムの作成段階で消費電力を低減させるための最適化処理を行うことができる、命令列最適化装置を提供する。

【構成】　プログラムを記憶するプログラムメモリとこのプログラムメモリから命令バスを介して前記プログラムを取り込む演算処理部とを備えた情報処理装置が使用するための前記プログラムを最適化する命令列最適化装置において、前記プログラムを構成する各命令について、相互の依存関係を解析する命令列解析手段と、この命令列解析手段で解析された依存関係に影響を与えない範囲で前記命令の順序を変更することによって、この命令を前記プログラムメモリから前記演算処理部に転送する際に前記命令バスに現れるビット列間のハミング距離を低減させる命令列変更手段とを備える。

1

【特許請求の範囲】
【請求項１】プログラムを記憶するプログラムメモリとこのプログラムメモリから命令バスを介して前記プログラムを取り込む演算処理部とを備えた情報処理装置が使用するための前記プログラムを最適化する命令列最適化装置において、
前記プログラムを構成する各命令について、相互の依存関係を解析する命令列解析手段と、
この命令列解析手段で解析された依存関係に影響を与えない範囲で前記命令の順序を変更することによって、この命令を前記プログラムメモリから前記演算処理部に転送する際に前記命令バスに現れるビット列間のハミング距離を低減させる命令列変更手段と、
を備えたことを特徴とする命令列最適化装置。
【請求項２】前記プログラムを基本ブロックに分割して分割後の基本ブロックを前記命令列解析手段に送るブロック分割手段をさらに備えたことを特徴とする請求項１記載の命令列最適化装置。
【請求項３】前記ブロック内命令列変更手段が、直前に命令順序決定処理を行った基本ブロックの最後の前記ビット列と今回命令順序決定処理を行う基本ブロックの最初の前記ビット列との間のハミング距離を考慮して、このブロック内での命令順序決定処理を行うことを特徴とする請求項２記載の命令列最適化装置。
【請求項４】前記命令が、前記演算処理部によって前記プログラムが実行される際に考慮されない前記ビット列を含む場合に、このビット列に前後するビット列とのハミング距離が低減されるように、このビット列の信号値を変更することを特徴とする請求項１～３のいずれかに記載の命令列最適化装置。
【請求項５】データを一時的に記憶する複数のレジスタと、プログラムを記憶するプログラムメモリと、このプログラムメモリから命令バスを介して取り込んだ命令にしたがって前記レジスタに対するデータの書き込み／読み出しを行う演算処理部とを備えた情報処理装置が使用するためのプログラムを最適化する命令列最適化装置において、
前記プログラムを構成する各命令中ののレジスタ番号を認識するレジスタ番号認識手段と、
このレジスタ番号認識手段で認識された前記レジスタ番号の有効範囲を認識するレジスタ有効範囲認識手段と、
このレジスタ有効範囲認識手段が認識した前記有効範囲に影響を与えない範囲で前記レジスタ番号を変更することによって、このレジスタ番号を含む命令を前記プログラムメモリから前記演算処理部に転送する際に前記命令バスに現れるビット列間のハミング距離を低減させる命令列変更手段と、
を備えたことを特徴とする命令列最適化装置。
【請求項６】前記命令列変更手段が、
前記レジスタ番号認識手段が認識した各レジスタ番号について、前記レジスタ有効範囲認識手段が認識した前記有効範囲に影響を与えることなく置き換えることができるレジスタ番号を策定する策定手段と、
前記レジスタ番号認識手段が認識したレジスタ番号および前記策定手段が策定したレジスタ番号のうち、前記命令バスに現れるビット列間のハミング距離が最も小さくなるレジスタ番号を選択する選択手段と、
前記プログラム中のレジスタ番号を前記選択手段が選択したレジスタ番号と置き換える置換手段と、
を備えたことを特徴とする請求項５記載の命令列最適化装置。
【請求項７】プログラムを記憶するプログラムメモリとこのプログラムメモリから命令バスを介して前記プログラムを取り込む演算処理部とを備えた情報処理装置が使用するための前記プログラムを最適化する命令列最適化装置において、
前記プログラムを構成する各命令の一部または全部について、同じ命令を意味する他のビットパターンを記憶する記憶手段と、
前記プログラム中の命令を前記記憶手段に記憶されたビットパターンに置換えることによって、この命令を前記プログラムメモリから前記演算処理部に転送する際に前記命令バスに現れるビット列間のハミング距離を低減させる命令列変更手段と、
を備えたことを特徴とする命令列最適化装置。
【請求項８】プログラムを記憶するプログラムメモリとこのプログラムメモリから命令バスを介して前記プログラムを取り込む演算処理部とを備えた情報処理装置が使用するための前記プログラムを最適化する命令列最適化装置において、
前記プログラム中の命令または命令列について、同じ処理結果を得ることができる他の命令または命令列を選定する選定手段と、
前記プログラム中の命令または命令列を前記選定手段で選定された命令または命令列と置換えることによって、この命令または命令列を前記プログラムメモリから前記演算処理部に転送する際に前記命令バスに現れるビット列間のハミング距離を低減させる命令列変更手段と、
を備えたことを特徴とする命令列最適化装置。
【請求項９】プログラムを記憶するプログラムメモリとこのプログラムメモリから命令バスを介して前記プログラムを取り込む演算処理部とを備えた情報処理装置が使用するための前記プログラムを最適化する命令列最適化装置において、
前記プログラム中の命令または命令列について、同じ処理結果を得ることができる他の命令または命令列を選定する選定手段と、
前記プログラム中の命令または命令列および前記選定手段で選定された命令または命令列について、これらの命令または命令列を前記プログラムメモリから前記演算処

理部に転送する際の前記命令バスにおける消費電力を試算する演算手段と、
前記選定手段で選定された命令または命令列のうち、前記演算手段で試算された消費電力が前記プログラム中の命令または命令列よりも小さいものを、このプログラム中の命令または命令列と置換える命令列変更手段と、
を備えたことを特徴とする命令列最適化装置。

【発明の詳細な説明】

【０００１】

【産業上の利用分野】本発明は、例えば情報処理装置の制御プログラム等を最適化するための命令列最適化装置に関するものである。

【０００２】

【従来の技術】近年のマルチメディアの発展に伴って、例えばブックコンピュータ、ノートコンピュータ、携帯電話等の携帯型情報処理装置が大きな普及を見せている。

【０００３】このような携帯型情報処理装置の制御部の概略構成を、図２６に示す。同図に示したＣＰＵ(Central Processing Unit) ２６１０において、実行ユニット２６１１は、制御プログラムを構成する各命令を実行する。また、入出力部２６１２は、実行ユニット２６１１が実行する命令のアドレスを順次アドレスバス２６３１に出力するとともに、このアドレスに対応する命令を命令バス２６３２から取り込む。レジスタ部２６１３は、実行ユニット２６１１による命令の実行に伴って生じたデータを一時的に記憶する。

【０００４】一方、プログラムメモリ２６２０において、記憶部２６２１には、制御プログラムを構成する各命令が予め記憶されている。また、入出力部２６２２は、アドレスバスから入力されたアドレスに対応する命令を記憶部２６２１から読み出して、命令バス２６３２に出力する。

【０００５】このような構成において、ＣＰＵ２６１０による制御を行う際には、まず、実行ユニット２６１１が実行する命令のアドレスが、アドレスバス２６３１を介して、ＣＰＵ２６１０の入出力部２６１２からプログラムメモリ２６２０の入出力部２６２２に送られる。これにより、プログラムメモリ２６２０は、指定されたアドレスに対応する命令を記憶部２６２１から読み出し、入出力部２６２２から出力する。ＣＰＵ２６１０の入出力部２６１２は、この命令を命令バス２６３２を介して入力し、実行ユニット２６１１に送る。そして、実行ユニット２６１１がこの命令を実行することによって、情報処理装置の制御を行っている。

【０００６】このようにして、実行ユニット２６１１が命令の実行を行っているときに、一時的に記憶する必要のあるデータが生じた場合、このデータはＣＰＵ２６１０内のレジスタ部２６１３に格納されるとともに、必要に応じて読み出される。

【０００７】ここで、図２６に示したような情報処理装置に使用する制御プログラムは、予め作成されて、情報処理装置の製造時に、プログラムメモリ２６２０に格納される。かかる制御プログラムの開発においては、高級言語或いはアセンブリ言語のコンパイル等を行って目的プログラム（すなわちプログラムメモリ２６２０に格納するプログラム）を作成する手段において、様々な最適化処理が施される。この最適化処理としては、例えば、制御プログラムの実行時間を短縮するための最適化処理や、この制御プログラムの格納に使用されるメモリ領域を低減させるための最適化処理などが、すでに知られている。

【０００８】

【発明が解決しようとする課題】上述したような情報処理装置においては、従来より、消費電力の低減が要求されている。特に、携帯型の情報処理装置では、連続使用できる時間を向上させるために、消費電力の低減が要求されている。また、携帯型以外の情報処理装置においても、環境保全やエネルギー消費削減の観点から、消費電力の低減が要求されている。

【０００９】情報処理装置の内部回路の消費電力Ｐは、以下のような式で表される。

【００１０】$P=\alpha \cdot C \cdot Vdd^2 \cdot n \cdot f+Ps$

ここで、αは稼働率、Ｃは回路全体のキャパシタンス、Ｖddは電源電圧、ｎは回路の素子数、ｆは動作周波数、Ｐｓ は待機時の消費電力である。

【００１１】従来は、これらの各パラメータの値が小さくなるようにハードウエアを構成することによって、消費電力の低減が図られていた。

【００１２】しかしながら、このようなハードウエア上の措置だけでは、消費電力を十分に低減させることはできなかった。

【００１３】本発明は、このような従来技術の欠点に鑑みてなされたものであり、情報処理装置用制御プログラムの作成段階で消費電力を低減させるための最適化処理を行うことができる、命令列最適化装置を提供することを目的とする。

【００１４】

【課題を解決するための手段】

（１）第１の発明に係わる命令列最適化装置は、プログラムを記憶するプログラムメモリとこのプログラムメモリから命令バスを介して前記プログラムを取り込む演算処理部とを備えた情報処理装置が使用するための前記プログラムを最適化する命令列最適化装置において、前記プログラムを構成する各命令について、相互の依存関係を解析する命令列解析手段と、この命令列解析手段で解析された依存関係に影響を与えない範囲で前記命令の順序を変更することによって、この命令を前記プログラムメモリから前記演算処理部に転送する際に前記命令バスに現れるビット列間のハミング距離を低減させる命令列

変更手段と、を備えたことを特徴とする。

（2）第2の発明に係わる命令列最適化装置は、データを一時的に記憶する複数のレジスタと、プログラムを記憶するプログラムメモリと、このプログラムメモリから命令バスを介して取り込んだ命令にしたがって前記レジスタに対するデータの書き込み／読み出しを行う演算処理部とを備えた情報処理装置が使用するためのプログラムを最適化する命令列最適化装置において、前記プログラムを構成する各命令中ののレジスタ番号を認識するレジスタ番号認識手段と、このレジスタ番号認識手段で認識された前記レジスタ番号の有効範囲を認識するレジスタ有効範囲認識手段と、このレジスタ有効範囲認識手段が認識した前記有効範囲に影響を与えない範囲で前記レジスタ番号を変更することによって、このレジスタ番号を含む命令を前記プログラムメモリから前記演算処理部に転送する際に前記命令バスに現れるビット列間のハミング距離を低減させる命令列変更手段と、を備えたことを特徴とする。

（3）第3の発明に係わる命令列最適化装置は、プログラムを記憶するプログラムメモリとこのプログラムメモリから命令バスを介して前記プログラムを取り込む演算処理部とを備えた情報処理装置が使用するための前記プログラムを最適化する命令列最適化装置において、前記プログラムを構成する各命令の一部または全部について、同じ命令を意味する他のビットパターンを記憶する記憶手段と、前記プログラム中の命令を前記記憶手段に記憶されたビットパターンに置換えることによって、この命令を前記プログラムメモリから前記演算処理部に転送する際に前記命令バスに現れるビット列間のハミング距離を低減させる命令列変更手段と、を備えたことを特徴とする。

（4）第4の発明に係わる命令列最適化装置は、プログラムを記憶するプログラムメモリとこのプログラムメモリから命令バスを介して前記プログラムを取り込む演算処理部とを備えた情報処理装置が使用するための前記プログラムを最適化する命令列最適化装置において、前記プログラム中の命令または命令列について、同じ処理結果を得ることができる他の命令または命令列を選定する選定手段と、前記プログラム中の命令または命令列を前記選定手段で選定された命令または命令列と置換えることによって、この命令または命令列を前記プログラムメモリから前記演算処理部に転送する際に前記命令バスに現れるビット列間のハミング距離を低減させる命令列変更手段と、を備えたことを特徴とする。

（5）第5の発明に係わる命令列最適化装置は、プログラムを記憶するプログラムメモリとこのプログラムメモリから命令バスを介して前記プログラムを取り込む演算処理部とを備えた情報処理装置が使用するための前記プログラムを最適化する命令列最適化装置において、前記プログラム中の命令または命令列について、同じ処理結果を得ることができる他の命令または命令列を選定する選定手段と、前記プログラム中の命令または命令列および前記選定手段で選定された命令または命令列について、これらの命令または命令列を前記プログラムメモリから前記演算処理部に転送する際の前記命令バスにおける消費電力を、ハミング距離を考慮して試算する演算手段と、前記プログラム中の命令または命令列を前記選定手段で選定された命令または命令列と置換えることによって、前記演算手段が試算した消費電力を低減させる命令列変更手段と、を備えたことを特徴とする。

【0015】

【作用】

（1）第1の発明に係わる命令列最適化装置によれば、プログラムを構成する各命令について相互の依存関係を解析し、この依存関係に影響を与えない範囲で命令の順序を変更することによってこの命令をプログラムメモリから演算処理部に転送する際に命令バスに現れるビット列間のハミング距離を低減させることとしたので、情報処理装置の消費電力を低減させることができる。

（2）第2の発明に係わる命令列最適化装置によれば、プログラムを構成する各命令のレジスタ番号を認識し、続いて、このレジスタ番号の有効範囲を認識し、そして、この有効範囲に影響を与えない範囲でレジスタ番号を変更することによって、このレジスタ番号を含む命令が転送される際に命令バスに現れるビット列間のハミング距離を低減させることとしたので、情報処理装置の消費電力を低減させることができる。

（3）第3の発明に係わる命令列最適化装置によれば、同じ命令を意味する他のビットパターンをプログラム中の命令と置換えることによって、この命令をプログラムメモリから演算処理部に転送する際に命令バスに現れるビット列間のハミング距離を低減させることとしたので、情報処理装置の消費電力を低減させることができる。

（4）第4の発明に係わる命令列最適化装置によれば、同じ処理結果を得ることができる他の命令または命令列をプログラム中の命令または命令列と置換えることによって、この命令または命令列をプログラムメモリから演算処理部に転送する際に命令バスに現れるビット列間のハミング距離を低減させることとしたので、情報処理装置の消費電力を低減させることができる。

（5）第5の発明に係わる命令列最適化装置によれば、プログラム中の命令または命令列、および、この命令または命令列と同じ処理結果を得ることができる他の命令または命令列の消費電力を試算し、この消費電力が小さい命令または命令列に置換することとしたので、情報処理装置の消費電力を低減させることができる。

【0016】

【実施例】以下、本発明の実施例について、図面を用いて説明する。

【００１７】（実施例１）実施例１として、第１の発明の一実施例（請求項１～４に対応する）について説明する。

【００１８】図１は、第１の発明（請求項１に対応する）の概念を概略的に示すフローチャートである。同図に示したように、第１の発明においては、まず、プログラムを構成する各命令について、命令列解析手段を用いて、相互の依存関係を解析する（ステップＳ１００）。そして、この依存関係に影響を与えることなく命令バス上に現れるビット列間のハミング距離が低減されるように、命令の順序を変更する（ステップＳ１０１）。そして、この命令順序の変更により、命令バスにおける消費電力の低減が実現される。

【００１９】このステップＳ１０１では、まず、消費電力低減化手段（第１の発明の「命令列変更手段」に相当する）を用いて、命令順序の変更と、このときのハミング距離の判定とを行う（ステップＳ１０２）。次に、判定されたハミング距離を所定の基準値と比較する（ステップＳ１０３）。ここで、基準値は、予め定められた値でもよいし、それまでに消費電力低減化手段で判定されたハミング距離の最低値であってもよい。そして、消費電力低減化手段で判定されたハミング距離が基準値よりも小さい場合は、この命令の順序を最適化の結果として出力し、この最適化処理を終了する（ステップＳ１０４）。一方、判定されたハミング距離が基準値よりも大きい場合は、他の命令順序について、同様の処理（ステップＳ１０２，Ｓ１０３）を繰り返す。

【００２０】次に、図１を具体化した例（請求項１～３に対応する）について、図２のフローチャートを用いて説明する。

【００２１】なお、ここでは、実行ユニットおよび命令バスがともに３２ビットの情報処理装置（図１４（ａ）参照）で使用される制御プログラムの最適化に第１の発明を適用した場合を例にとって説明する。

【００２２】図２において、ステップＳ２００では、制御プログラムを、基本ブロックに分割する。ここで、基本ブロックとは、例えば式や代入文の並びのような、途中から外部への分岐が起こらず、また、外部から途中への分岐も起こらないプログラムブロックをいう。基本ブロックへの分割が終了すると、次に、この基本ブロック内の各命令シーケンスに対して、それらの命令の依存関係およびレジスタの依存関係を解析し、命令の入れ換えによって処理の因果律を侵さない範囲を特定する。そして、このようにして特定された各範囲を識別するための識別子をプログラムに付加する。

【００２３】ステップ２００における処理は、従来から知られているような他の最適化処理においても利用される処理であるので、ここでは詳細の説明を省略する。このような処理を開示した文献としては、例えば以下のようなものがある。

【００２４】Z.Li and P-C.Yew, " Efficient Interprocedual Analysis fir Program Parallelization and Restructuring," Proc.ACM SIGPLAN PPEALS,pp85-97,1988.S.Jain and C.Thompson," An Efficient Approach to Dataflow Analysis ina Multiple Pass Global Optimizer, " Proc.SIGPLAN' 88 Cof.on Prog.Lang.Design and Implementation(PLDI' 88),pp.154-163,1988.

次に、ステップＳ２０１で、基本ブロックごとに最適化処理を行う。以下、このステップＳ１０１の処理手順について説明する。

【００２５】まず、ステップＳ２０２により、初期設定を行う。図２において、変数ＬａｓｔＣｏｍは、前回に最適化処理を行った基本ブロックの最後の命令が代入されている。初期設定においては、この変数ＬａｓｔＣｏｍに、デフォルト値（命令ビット列のひとつ）を代入する。このデフォルト値としては、どのような値を使用してもよく、例えば、全ビット“１”や全ビット“０”であってもよい。但し、現実的には、統計的に最も頻繁に出現する命令や、最終的にリンクされるヘッダ・プログラム（プロローグ・プログラム、すなわち、ＯＳから目的のユーザが作成したプログラムを起動させ、実行終了後のＯＳに戻るためのプログラム）或いはランタイム・ルーチンなどの最終命令などをデフォルト値とすることが好ましい。

【００２６】以降のステップＳ２０３～Ｓ２１２では、ハミング距離を最小にするための処理が行われる。ここで示す手順は、命令の順番を順次変更し、可能な全パターンを試行することによって最適化を行う、素朴且つ確実な「しらみつぶし」による方法を用いている。なお、複雑なデータ構造の基本ブロックの場合は、特別な手法を用いて高速処理を行うことも有効であるが、本発明では特に限定されるものではない。

【００２７】ステップＳ２０３においては、全ての処理が終了したか否かの判定を行う。本実施例における最適化処理は、基本ブロックごとに行われるので、すべての基本ブロックに対する処理が終了した時点で終了する。この判定の結果、処理を行っていない基本ブロックが残っている場合には、ステップＳ２０４以降の処理を実行する。

【００２８】ステップＳ２０４では、基本ブロック内の最適化処理が終了したか否かの判定を行う。この最適化処理は、基本ブロック内の各命令の実行順序を置き換えることによって行われる。注目している基本ブロックに対する処理は、命令の順序依存やレジスタ依存などに矛盾を生じさせない全ての可能な置き換え方（順列）について試行し終わっている場合に終了し、ステップＳ２１３に進む。一方、全ても置き換え方の試行を終了していない場合は、ステップＳ２０５以降の最適化処理を続行する。

【００２９】ステップＳ２０５では、ブロック内の最適

化に先立つ初期設定を行う。図2において、変数Ｈｄ＿ｓｕｍは、注目している基本ブロック内での命令間のハミング距離の総和を示し、変数Ｈｄ＿ｂｏｕｎは変数ＬａｓｔＣｏｍと注目している基本ブロックの先頭命令とのハミング距離である。また、変数Ｈｄ＿ｔｏｔａｌは、変数Ｈｄ＿ｓｕｍと変数Ｈｄ＿ｂｏｕｎとの和を示し、変数Ｈｄ＿ｍｉｎは、変数Ｈｄ＿ｔｏｔａｌの最小値を示している。この変数Ｈｄ＿ｍｉｎに、初期設定として、“∞”を代入する。ここで、“∞”はいかなる数字よりも大きい値であるものとする。

【００３０】ステップＳ２０６では、注目している基本ブロック内のハミング距離の総和を求め、変数Ｈｄ＿ｓｕｍに代入する。この総和は、隣接する命令のビット・パターンを比較し、対応する桁のビットの値が異なっている場合を“１”として累積加算することにより簡単に求めることができる。

【００３１】ステップＳ２０７では、ステップ２０６と同様にして、前回処理した基本ブロックの最後の命令（変数ＬａｓｔＣｏｍ）と今回注目している基本ブロックの先頭命令とのハミング距離を求め、変数Ｈｄ＿ｂｏｕｎに代入する。この操作を行うことによって、基本ブロック間にまたがる最適化処理が可能となる。

【００３２】ステップＳ２０８では、現在の命令シーケンス・パターンにおける命令間のハミング距離の総和、すなわち変数Ｈｄ＿ｓｕｍと変数Ｈｄ＿ｂｏｕｎとの和を算出して、変数Ｈｄ＿ｔｏｔａｌに代入する。

【００３３】ステップＳ２０９では、ステップＳ２０８で得られたＨｄ＿ｔｏｔａｌの値と、現在までの試行によって得られたＨｄ＿ｔｏｔａｌのうちの最小値であるＨｄ＿ｍｉｎとが、比較される。そして、Ｈｄ＿ｔｏｔａｌ≧Ｈｄ＿ｍｉｎの場合は、異なる命令シーケンス・パターンについての試行をさらに行うべく、ステップＳ２１２以降を実行する。一方、Ｈｄ＿ｔｏｔａｌ＜Ｈｄ＿ｍｉｎの場合は、ステップＳ２１０，Ｓ２１１を実行した後、ステップＳ２１２以降を実行する。

【００３４】ステップＳ２１０，Ｓ２１１では、変数の更新を行う。まず、ステップＳ２１０においては、Ｈｄ＿ｔｏｔａｌの値をＨｄ＿ｍｉｎに代入する。また、ステップＳ２１１においては、このＨｄ＿ｍｉｎに対応する命令シーケンス・パターンを、変数ＭｉｎＨｄＳｅｑｕｅｎｃｅに記憶する。

【００３５】ステップＳ２１２においては、命令シーケンスの入れ換えを行う。ここでは、基本ブロック内の命令の順番を入れ換えることにより、まだ試行していない命令シーケンスを生成して、ステップＳ２０４に戻る。この命令の入れ換えにおいては、ステップＳ２００で行った解析の結果に基づいて、因果律に矛盾が生じないようにする。

【００３６】以上説明したステップＳ２０３～Ｓ２１２からなる処理を反復して行うことにより、最適化を行うことができる。

【００３７】すべての命令シーケンス・パターンについての試行を終了すると（ステップＳ２０４）、続いて、ステップＳ２１３を実行する。ステップＳ２１３では、最適化された命令シーケンス（変数ＭｉｎＨｄＳｅｑｕｅｎｃｅに記憶されている）を、最適化の結果として出力する。

【００３８】ステップＳ２１４では、変数ＬａｓｔＣｏｍの更新を行う。すなわち、この変数ＬａｓｔＣｏｍに、変数ＭｉｎＨｄＳｅｑｕｅｎｃｅに記憶された最後の命令を代入する。

【００３９】そして、以上の説明と同様にして、次の基本ブロックについて、ステップＳ２０３以降の処理を実行する。

【００４０】次に、本実施例の最適化処理を実際に行う場合について、図3に示したようなプログラムを用いる場合を例にとって説明する。

【００４１】図3のプログラムは、整数の内積演算を行うためのプログラムであり、Ｃ言語で記述されている。

【００４２】図4～図7は、図3に示したプログラムを“Ｓｕｎ　ＳＰＡＲＣ　Ｃコンパイラ”でコンパイルした場合のアセンブリ・ソース・プログラムリストを示している。図4～図7において、第1カラムはライン番号、第2カラムはアドレス、第3カラムはオブジェクト・コード、第4カラムはアセンブリ・ソースを、それぞれ示している。なお、Ｃ１０１～Ｃ３１３およびＢ１～Ｂ６は、説明のための符号である。ここで、アセンブリ文法やニーモニック等については、公知技術であるので説明を省略するが、これらを開示した文献としては例えば以下のようなものがある。

【００４３】SPARC International,Inc.,The SPARC Architecture Manual Version 8,Prectice-Hall,Inc.A Simon and Schuster Company Englewood Cliffs,New Lersey 07632.

本実施例の最適化処理装置に入力されるのは、図4～図7に示したようなアセンブリ・ソースである。そして、このアセンブリ・ソースは、図2にステップＳ２００によって基本ブロックが検索される。これにより、このプログラムは、図4～図7にＢ１～Ｂ６で示したような6個の基本ブロックに分割される。

【００４４】さらに、各基本ブロック内の命令の依存関係が解析される。図8および図9は、依存関係の解析結果を示す有向グラフである。ここで、図8は基本ブロックＢ１（図4参照）の有向グラフであり、図4のＣ１０１～Ｃ１０４はそれぞれ図8のノードＮ１０１～Ｎ１０４に対応している。同様に、図9は基本ブロックＢ３（図5参照）の有向グラフであり、図5のＣ３０１～Ｃ３１３はそれぞれ図9のノードＮ３０１～Ｎ３１３に対応している。また、ノードＮ１００，Ｎ３００は有向グラフのトップを表すダミー・ノードであり、ノードＮ１

０５，Ｎ３１４は有向グラフのボトムを表すダミー・ノードである。さらに、アークＡ１０１～Ａ１０５，Ａ３０１～Ａ３１４は命令の依存関係を示し、矢印の逆の順番に命令を実行してはならないことを示している。基本ブロック内の命令の順序関係は、このような有向グラフを作成することによって管理する。

【００４５】まず、第1の基本ブロックＢ１の最適化処理について説明する。基本ブロックＢ１は、本プログラムを実行するために内部状態を保存するためのプロローグ処理である。図8の有向グラフに示したように、４個の命令Ｃ１０１～Ｃ１０４は、この順番でしか実行できないので、そのまま出力される。

【００４６】第2の基本ブロックＢ２の最適化処理については説明を省略するが、この基本ブロックＢ２の処理の終了時には、最後の命令Ｃ２０４としての“ｎｏｐ”がＬａｓｔＣｏｍに記憶されている。

【００４７】次に、第3の基本ブロックＢ３の最適化処理を行う。基本ブロックＢ３においては、図9の有向グラフに示したように、“Ｃ３０１”、“Ｃ３０２～Ｃ３０４”、“Ｃ３０５”、“Ｃ３０６～Ｃ３０８”相互間で命令シーケンスの順序を入れ換えることは可能であるが、“Ｃ３０２～Ｃ３０４”や“Ｃ３０６～Ｃ３０８”内で命令の順序を入れ換えることはできない。また、Ｃ３０９～Ｃ３１３は、この順番でしか実行できず、Ｃ３０１～Ｃ３０８よりも先に実行することはできない。

【００４８】まず、命令シーケンス・パターンを図5のとおり、すなわちＣ３０１，Ｃ３０２・・・Ｃ３１３とした場合について、図2に示したような最適化処理ステップＳ２０５～Ｓ２１１を行う。このとき、ＬａｓｔＣｏｍには、上述したように、基本ブロックＢ２の最後の命令Ｃ２０４としての“ｎｏｐ”が記憶されている。また、最小ハミング距離Ｈｄ＿ｍｉｎには、初期設定として“∞”が代入されている。

【００４９】図１０は、基本ブロックＢ２の最後の命令Ｃ２０４および基本ブロックＢ３の各命令Ｃ３０１～Ｃ３１３のビット・パターンを示している。このようなビット・パターンについて、Ｈｄ＿ｓｕｍ，Ｈｄ＿ｂｏｕｎ，Ｈｄ＿ｔｏｔａｌを算出すると（図2のステップＳ２０６～Ｓ２０８参照）、Ｈｄ＿ｓｕｍ＝１６１、Ｈｄ＿ｂｏｕｎ＝１３となり、したがってＨｄ＿ｔｏｔａｌ＝１７４となる。

【００５０】ここで、最小ハミング距離Ｈｄ＿ｍｉｎ＝∞であるので、ステップＳ２０９（図2参照）での比較の結果、Ｈｄ＿ｍｉｎにはＨｄ＿ｔｏｔａｌ＝１７４が代入され（ステップＳ２１０）、さらに、図5に示したような命令シーケンスが変数ＭｉｎＨｄＳｅｑｕｅｎｃｅに記憶される（ステップＳ２１１）。

【００５１】次に、命令の順序を入れ換えた場合について、同様の処理（ステップＳ２０５～Ｓ２１１）を行う。

【００５２】そして、入れ換えが可能な命令シーケンスのすべてについての試行が終了すると、変数ＭｉｎＨｄＳｅｑｕｅｎｃｅに記憶されている命令シーケンスを、命令シーケンスの最適化の結果として出力する（ステップＳ２１３）。

【００５３】ステップＳ２１３で出力された命令シーケンスのビットパターンを図１１に示す。また、参考例として、最悪の（すなわち、ハミング距離の総和が最大になる）ビットパターンを図１２に示す。図１１におけるハミング距離の総和Ｈｄ＿ｔｏｔａｌは１３０となる。また、図１２におけるハミング距離の総和Ｈｄ＿ｔｏｔａｌは１９６となる。すなわち、本実施例によれば、最適化処理によって、基本ブロックＢ３を実行する際の命令バスのスイッチング回数を最適化前の７４．７％とすることができ、また、最悪の場合の６６．３％とすることができた。

【００５４】以下、同様にして第４～第６の基本ブロックＢ４～Ｂ６についての最適化処理を行うが、これらの各ブロックＢ４～Ｂ６内では命令の順番を入れ換えることはできないので、そのまま出力して処理を終了する。

【００５５】次に、本実施例に係わる命令列最適化装置の変形例（請求項４に対応する）について、図１３を用いて説明する。

【００５６】プログラム中の命令によっては、命令フォーマットの中に“do not care”のビット、すなわち“１”あるいは“０”のどちらであっても、その命令の動作に影響を与えないビットを含む場合がある。例えば、上述の基本ブロックＢ３において、命令Ｃ３０３，Ｃ３０４，Ｃ３０７，Ｃ３０８の１２ビット目から６ビット目（ビット〈１１：５〉）は、“do not care”のビットである（図１０参照）。このようなビットの値を適当に変更することによって、隣接する命令間でのハミング距離を低減させることができる場合がある。

【００５７】図１３は、“do not care”のビットの値を変更することによってハミング距離を低減させるための処理の一例を示すフローチャートである。

【００５８】図１３に示したような処理を、図2のステップＳ２０６に換えて実行することにより、“do not care”のビットを考慮して、さらなるハミング距離の低減を図ることができる。

【００５９】同図において、ステップＳ１３０１では、初期設定として、変数Ｈｄ＿ｓｕｍに初期値“∞”を代入する。

【００６０】次に、ステップＳ１３０２において、本処理が終了したか否かの判定を行う。本処理においては、“do not care”の全ビットの値を変更しつつ、以下のような試行を行う。そして、“do not care”のビットの“１”，“０”の組み合わせについてのすべての試行を終了すると、本処理を終了する。

【００６１】ステップＳ１３０３では、現時点での“do

not care ”のビット・パターンについて、隣接する命令間でのハミング距離の総和を求め、変数Ｈｄ＿ｓｕｍ＿ｃｕｒｒｅｎｔに代入する。

【００６２】ステップＳ１３０４では、Ｈｄ＿ｓｕｍとＨｄ＿ｓｕｍ＿ｃｕｒｒｅｎｔとの大小比較を行う。ここで、Ｈｄ＿ｓｕｍ≧Ｈｄ＿ｓｕｍ＿ｃｕｒｒｅｎｔであれば、ステップＳ１３０５でＨｄ＿ｓｕｍにＨｄ＿ｓｕｍ＿ｃｕｒｒｅｎｔの値を代入したのち、ステップＳ１３０６へ進む。一方、Ｈｄ＿ｓｕｍ＜Ｈｄ＿ｓｕｍ＿ｃｕｒｒｅｎｔであれば、ステップＳ１３０５を実行することなく、そのままステップＳ１３０６へ進む。

【００６３】ステップＳ１３０６では、“do not care ”のビット・パターンを、まだ試行していないビット・パターンに変更する。

【００６４】このような処理を、例えば基本ブロックＢ３の最適化前のプログラム（図１０参照）においては、ハミング距離の総和Ｈｄ＿ｓｕｍ，Ｈｄ＿ｔｏｔａｌを１０だけ低減させることができる。

【００６５】以上説明したようにして最適化を行った制御プログラムを情報処理装置のプログラムメモリに格納し、この制御プログラムを用いてＣＰＵ等の制御を行うことにより、命令バスにおける消費電力を低減させることが可能となる。

【００６６】本実施例に示した手順は、最適化を入力データに対して順次実行する１パス方式のものである。そのため、最終的に得られた命令例は、ハミング距離が最小でない場合もあり得る。これは、基本ブロックの境界の最適化において、注目している基本ブロックの一つ前の基本ブロックの最終命令しか考慮していないためである。したがって、ハミング距離をさらに低減させるためには、例えば、注目している基本ブロックの次の基本ブロックやさらに次の基本ブロック等をも考慮する方法等が考えられる。しかし、単純な処理で迅速に最適化を行うためには、基本ブロックに分割して処理を行う方が望ましい。

【００６７】また、アセンブリ・ソースに対して本実施例に係わる最適化処理と他の最適化処理（例えば、制御プログラムの実行時間を短縮するための最適化処理や、この制御プログラムの格納に使用されるメモリ領域を低減させるための最適化処理など）を行う場合には、各最適化処理を行う順番に係わらず、本実施例の効果を得ることができる。しかし、本実施例の効果を最も有効に得るためには、本実施例に係わる最適化を最後に行うことが望ましい。ここで、本実施例の最適化処理を最後に行う場合には、他の最適化処理の結果を変更してしまう場合が考えられるが、このような不都合の防止は、ステップＳ１００（図１参照）の依存解析のフェイズにおいて考慮すればよい。すなわち、図８、図９に示したような有向グラフにおいて、他の最適化処理の結果を変更しないように制約の設定を行えばよい。

【００６８】以上説明した本実施例では、実行ユニットおよび命令バスがともに３２ビットの情報処理装置で使用される制御プログラムの最適化を例にとった。すなわち、本実施例で最適化された制御プログラムは、図１４（ａ）に示したような、命令を一度に１個ずつ読み出して、フェッチ、発行、デコード、実行を行う情報処理装置の制御に使用されることを前提としていた。したがって、最適化処理において考慮すべきハミング距離は、図１５（ａ）に示すように、隣接する命令間のハミング距離である。

【００６９】しかしながら、図１４（ｂ）に示したように、今日ではＣＰＵの多くは一度に複数の命令の読み出し、フェッチ、発行が行える構成となっている。このようなＣＰＵを用いる場合、最適化を行う際に考慮すべきハミング距離は、隣接する命令間のハミング距離ではなく、命令バスの同じフィールド、同じビット位置に割り当てられる命令間のハミング距離である。すなわち、図１４（ｂ）に示したように、一度に２命令ずつ命令を読み出すような構成のＣＰＵの制御プログラムを最適化する場合は、図１５（ｂ）に示したように、１つおきの命令間（例えば、Ｃ３０１とＣ３０３、Ｃ３０２とＣ３０４等）のハミング距離が低減されるように最適化を行えばよい。このような場合、２個づつの命令をビット結合（コンカチネント）して得られたビット列を作成することとすれば、本実施例の最適化装置をそのまま用いて最適化処理を行うことができる。

【００７０】図１６（ａ）は、一度に４命令ずつ命令を読み出すような構成の装置を示している。このような場合も、図１６（ｂ）に示したように、３つおきの命令間（例えば、Ｃ３０１とＣ３０５、Ｃ３０２とＣ３０６等）のハミング距離が低減されるように最適化を行えばよい。そして、４個づつの命令をビット結合して得られたビット列を作成することとすれば、本実施例の最適化装置をそのまま用いて最適化処理を行うことができる。

【００７１】また、情報処理装置によっては、内部の命令バスのバンド幅と外部の命令バスのバンド幅とが異なる場合がある。図１７は、内部の命令バスのバンド幅は１２８ビットであるが、外部の命令バスのバンド幅は３２ビットである場合を示している。このような場合には、内部バスについては３命令おきの命令間でハミング距離の低減を行い、外部バスについては隣接する命令間でハミング距離の低減を行えばよい。どちらを優先するか、或いは互いに妥協するのかは、消費電力の低減に対する関与の度合い等に応じて、適宜決定すればよい。

【００７２】（実施例２）実施例２として、第２の発明の一実施例（請求項５、６に対応する）について説明する。

【００７３】なお、ここでは、実行ユニットおよび命令バスがともに３２ビットの場合を例にとって説明する。

【００７４】本実施例の命令列最適化装置においては、

レジスタに割り当てられるべき変数に注目して、制御プログラムの最適化を行う。すなわち、その変数が現れる命令列のある区間でビットの変化量を見て、最小となるレジスタ番号を割り当てていく。

【００７５】本実施例では、ｃ＝ａ－ｂ、ｃ＝ａ／ｂ等において、ａをソース、ｂをターゲット、ｃをデスティネーションとし、ａの値を保持するレジスタをソース・レジスタ、ｂの値を保持するレジスタをターゲット・レジスタ、ｃの値を保持するデスティネーション・レジスタと称することとする。そして、３２ビットの命令のうち、左側ＭＳＢから数えて、１ビット目～１０ビット目および２１ビット目～２７ビット目を命令コードのフィールドとし、１１ビット目～１５ビット目をデスティネーション・レジスタのフィールドとし、１６ビット目～２０ビット目をソース・レジスタのフィールドとし、２７ビット目～３２ビット目をターゲットレジスタのフィールドとする。

【００７６】あるレジスタ番号に格納されるデータの有効範囲は、このレジスタ番号がデスティネーションのレジスタ番号として現れる命令によってデータがレジスタに格納されてから、この格納データが必要とされる命令、すなわち、このレジスタ番号がソースレジスタ或いはターゲットレジスタとして現れる命令までである。当然ながら、一つのレジスタ番号は複数の変数或いはデータの一時記憶場所として使い回しをして効率化を図っている。したがって、プログラム中で、１個のレジスタ番号がある一つのレジスタデータを保管している範囲を知るには、コンパイラの最適化部でレジスタ番号を割り当てる際に作られるレジスタ割当テーブルを解析し、有効範囲テーブルを作製することが必要となる。

【００７７】有効範囲テーブルは、レジスタ割当テーブルから容易に作製することができる。通常、コンパイラでは、ソースプログラムからデータフローグラフ或いは依存グラフを生成する。そして、このグラフを用いて、ある変数のデータまたは一時的な演算の中間結果を示すデータの保持のためにレジスタを割り当てていく。ここで、従来は、変数が現れるとレジスタを割り当てテーブルに登録し、データフローグラフ或いは依存グラフから不要になったと判断された変数についてはレジスタの割当テーブルからエントリーを削除していた。そして、これにより、あるレジスタ番号のレジスタに保持されているデータが有効なアドレスの範囲、すなわち、あるいロード命令或いは演算結果の書き込みにより有効なデータが書き込まれてからそのデータを必要とする最後の演算命令或いはストア命令が出現するまでの有効期間を判断していた。

【００７８】レジスタ番号を割り当てる際に、例えば、

```
ｃ＝ａ＋ｂ
ｃ＝ｃ＊ｄ
```

というソースプログラムがあった場合、

ａ：レジスタ０
ｂ：レジスタ１
ｃ：レジスタ２
ｄ：レジスタ３

とし、コンパイル後のプログラムを

（１）

```
ａｄｄ　ｒ０，ｒ１，ｒ２
ｍｕｌ　ｒ２，ｒ３，ｒ２
```

としてもよいが、乗算結果としてのｃのみをレジスタ５に格納すること、すなわち、

（２）

```
ａｄｄ　ｒ０，ｒ１，ｒ２
ｍｕｌ　ｒ２，ｒ３，ｒ５
```

としてもよい。この場合には、ソースプログラムの変数ｃに対し、複数のレジスタ番号を割り当てることになる。従来は、上記プログラム（１）のように、例えばＳＰＡＲＣのレジスタウインドウのグローバルレジスタのようにレジスタに特殊な意味または機能がある場合を除いては、レジスタリソースの問題からレジスタからのデータの退避が必要ないときには複数のレジスタ番号への割り付けをできるだけ避けている。これに対して、本実施例では、上記プログラム（２）のように、複数のレジスタへの割り当てを行うものとする。これにより、評価対象の評価範囲を分割することができるので、評価範囲が狭まり、評価対象を増加させることができる。一方、上述のようにレジスタに特殊な意味・機能がある場合には、もともと割り当てられていたレジスタと同じ機能を有するレジスタ以外には割り当てることができないので、選択範囲が狭くなり、注意が必要である。

【００７９】有効範囲テーブルを作成した後は、注目するレジスタ番号についてハミング距離を求め、割り付け可能な他のレジスタ番号の再割り当てを行う。この再割り当てを行う際には、さらに複数のレジスタに割り当てることを試みることとして、評価範囲の分割を図ることも可能である。

【００８０】また、本実施例では、複数のレジスタ番号について同時に評価することも可能である。同時に評価を行うレジスタ番号を１つに限定すると、置き換えが可能なレジスタ番号の数が限定されるが、複数のレジスタ番号について同時に評価することにより、これらの複数のレジスタ番号を互いに置き換えが可能なレジスタ番号として置き換えの最適化を行うことができる。

【００８１】さらに、レジスタファイルの再割り当てを行う段階は、コンパイラでレジスタ割り当てを行うときでもよいし、一旦割り当てを行った後であってもよい。

【００８２】次に、本実施例の命令列最適化装置の具体的な例について、図１８～図２０を用いて説明する。

【００８３】図１８は、本実施例の命令列最適化装置が行う最適化処理の手順を説明するためのフローチャートである。

【００８４】まず、高級言語或いはアセンブリ言語で作製されたソース・プログラムをコンパイルし、さらに他の最適化処理を施すことにより、中間コード（アセンブリ・コード）を作製する（ステップＳ１８０１）。このようにして得られた中間コードのプログラム例を、図１９（ａ）に示す。

【００８５】次に、レジスタ割当テーブルを作製し、さらに、このレジスタ割当テーブルから有効範囲テーブルを作製する（ステップＳ１８０２）。図１９（ａ）に示したプログラムの有効範囲テーブルを、図２０に示す。

【００８６】そして、今回の試行で注目するデータを選択し（ステップＳ１８０３）、この注目データに割当られたレジスタ番号を、ハミング距離が最低となるようなレジスタ番号と置き換える（ステップＳ１８０４）。このとき、有効範囲の境界での命令のハミング距離も考慮し、この境界におけるハミング距離も小さくなるような最適化を行う。

【００８７】ここで、レジスタ番号０ｘ１ｃに割り当てられていたデータが注目データである場合について考える。このデータの有効範囲は、図２０の有効範囲テーブルより、アドレス０１０１からアドレス１０００までであることがわかる。すなわち、かかるデータについては、アドレス０１０１～１０００について評価すればよい。なお、図１９（ａ）のプログラムでは、アドレス１１０１の命令でレジスタ番号０ｘ１ｃに新たなデータが格納されているが、評価対象外なので評価しない。

【００８８】図１９（ａ）のプログラムでは、レジスタ番号０ｘ１ｃの前後でのハミング距離の総和は１４となっている。ここで、ハミング距離の総和を最小にする他のレジスタ番号を探すと、レジスタ番号０，２に置き換えることによってハミング距離の総和を８にできることがわかる。ここでは、レジスタ番号２については有効範囲が重複しているので、レジスタ番号０ｘ１ｃをレジスタ番号０に置き換えることとする。これにより、図１９（ｂ）に示したようなプログラムを得ることができる。

【００８９】プログラムの置き換えが終了すると、続いて、すべてのデータについて最適化が終了したか否かを判定する（ステップＳ１８０５）。そして、最適化が終了していないデータが残っている場合には、そのデータについてステップＳ１８０３～Ｓ１８０５を実行する。一方、すべてのデータについて最適化が終了している場合には、最適化処理後のプログラムを出力し、最適化処理を終了する。

【００９０】本実施例によれば、以上説明したようにして最適化を行った制御プログラムを情報処理装置のプログラムメモリに格納し、この制御プログラムを用いてＣＰＵ等の制御を行うことにより、命令バスにおける消費電力を低減させることが可能となる。

【００９１】なお、本実施例では、実行ユニットおよび命令バスがともに３２ビットの場合を例に採って説明したが、複数ワードのアドレスを同時に転送するような場合にも、第２の発明を適用できることはもちろんである。例えば４ワードバウンダリで転送する場合には、注目する命令の４ワード前の命令および４ワード後の命令とのハミング距離について評価を行えばよい。

【００９２】（実施例３）次に、実施例３として、第３の発明の一実施例（請求項７に対応する）について説明する。

【００９３】本実施例では、“ａｄｄ”命令に第３の発明を適用した場合を例に採って説明する。

【００９４】命令機能コードを作成するとき、例えばＳＰＡＲＣの命令機能コードを参照すると、“ａｄｄ”命令は“０００００”であるが、この“ａｄｄ”命令等は非常に出現頻度が高い命令であるので、本実施例では、“０００００”に加えて“１１１１１１”も“ａｄｄ”命令となるように、命令体系を作成する。すなわち、本実施例の命令列最適化装置で最適化された制御プログラムを使用する情報処理装置においては、“００００００”および“１１１１１１”を“ａｄｄ”命令であるとしてデコードするように、ＣＰＵの命令デコーダが構成されるものとする。これにより、コンパイラは、オブジェクト・コードを作製する際に、命令機能コードのフィールドに“０００００”または“１１１１１１”のいずれかを割り当てることができる。ここで、“ａｄｄ”命令の前後の命令の命令機能コードのフィールドが“００１１１０”および“１１０１１０”であったとすると、“ａｄｄ”命令の命令機能コードに“０００００”を選択した場合のハミング距離は７であり、“ａｄｄ”命令の命令機能コードに“１１１１１１”を選択した場合のハミング距離は５である。したがって、この場合には、コンパイラは、ａｄｄ”命令の命令機能コードに“１１１１１１”を選択する。

【００９５】次に、本実施例の命令列最適化装置の具体的な例について、図２１および図２２を用いて説明する。

【００９６】図２１は、本実施例の命令列最適化装置が行う最適化処理の手順を説明するためのフローチャートである。

【００９７】まず、高級言語或いはアセンブリ言語で作製されたソース・プログラムをコンパイルし、さらに他の最適化処理を施すことにより、中間コード（アセンブリ・コード）を作製する（ステップＳ２１０１）。このとき、“ａｄｄ”命令の命令機能コードは“０００００”となっているものとする。

【００９８】次に、各命令について、本実施例を適用する命令であるか否か、すなわち命令機能コードのフィールドを複数割り当てられている命令（ここでは“ａｄｄ”命令）であるか否かを、判断する（ステップＳ２１０２）。

【００９９】そして、本実施例を適用する命令であると

判断された場合は、この命令に対して、置換が可能なビットパターン（ここでは“１１１１１１”）を選出する（ステップＳ２１０３）。

【０１００】さらに、この命令に対応するビットパターンのすべてについて、その前後の命令とのハミング距離を算出し、互いに比較することによって、ハミング距離が最低となるようなビットパターンを選択する（ステップＳ２１０４）。図２２に、“ａｄｄ”命令についての最適化処理を行ったプログラムの例を示す。この例では、先に現れた“ａｄｄ”命令では命令機能コードを“０００００”とした方がハミング距離が小さいので置換を行わず、後に現れた“ａｄｄ”命令では命令機能コードを“１１１１１”とした方がハミング距離が小さいので置換を行っている。

【０１０１】プログラムの置換が終了すると、続いて、すべてのデータについて最適化が終了したか否かを判定する（ステップＳ２１０５）。そして、最適化が終了していないデータが残っている場合には、そのデータについてステップＳ２１０３～Ｓ２１０５を実行する。一方、すべてのデータについて最適化が終了している場合には、最適化処理後のプログラムを出力し、最適化処理を終了する。

【０１０２】本実施例によれば、以上説明したようにして最適化を行った制御プログラムを情報処理装置のプログラムメモリに格納し、この制御プログラムを用いてＣＰＵ等の制御を行うことにより、命令バスにおける消費電力を低減させることが可能となる。

【０１０３】（実施例４）次に、実施例４として、第４の発明の一実施例（請求項８に対応する）について説明する。

【０１０４】本実施例では、１種類の動作を行うための実現方法が複数ある場合に、前後の命令とのハミング距離が最も小さくなるように、その実現方法に係わる命令を選択する。例えば、レジスタ０ｘ０ｄにデータ“０”を書き込む場合、ＳＰＡＲＣのようにレジスタ番号０が書き込みは意味がないが読み出しはデータ“０”を出力する特別なレジスタとして定義されている場合、その実現方法としては、以下のようなものがある。これらの実現方法（すなわち命令）のうちで、前後の命令とのハミング距離が最も小さくなるものを選択して、その命令を置き換えることとする。

【０１０５】

ｍｏｖ　ｒ０，ｒｄ
（０ｘ０ｄにデータ“０”を移す命令）
ａｄｄ　ｒ０，ｒ０，ｒｄ
（０＋０を０ｘ０ｄに格納させる命令）
ｍｕｌ　ｒ？，ｒ０，ｒｄ
（ある値に０を掛けた値を０ｘ０ｄに格納させる命令）
ｍｕｌ　ｒ０，ｒ？，ｒｄ
（０にある値を掛けた値を０ｘ０ｄに格納させる命令）
ｘｏｒ　ｒ？，ｒ？，ｒｄ
（ある値と、これと同じ値との排他的論理和を取った結果を０ｘ０ｄに格納させる命令）
ｓｌｌ　ｒ０．ｒ？，ｒｄ
（０をある値だけ右にシフトさせた値を０ｘ０ｄに格納させる命令）
ｓｒｌ　ｒ０，ｒ？，ｒｄ
（０をある値だけ左にシフトさせた値を０ｘ０ｄに格納させる命令）

また、他の具体例としては、イミディエイト加算命令によるものがある。イミディエイト加算に対して、イミディエイト部分を２の補数としたイミディエイト減算は、演算機能としてはまったく同じである。例えば、

ａ＝ｂ＋５

と、

ａ＝ｂ－（－５）

とは、同じ演算として扱われる。ここで、イミディエイト加算とイミディエイト減算とを置き換えた場合、イミディエイトデータを表す命令のフィールドが反転するので、両式を置き換えることによってハミング距離を低減することができる場合がある。

【０１０６】本実施例において、置き換えを行う候補を選出するためには、例えば、いわゆるライブラリを予め作製しておき、ある命令を評価するときにこのライブラリに置き換え候補が登録されているか否かを検索すればよい。検索の結果、置き換え候補が検索された場合には、この置き換え候補を採用した場合とハミング距離を比較する。また、イミディエイト加算とイミディエイト減算とを置き換えるためには、ライブラリに、イミディエイト加算の置き換え候補として、イミディエイト減算を登録しておけばよい。このとき、イミディエイトデータを変換する方式或いは手順もライブラリに登録しておけば、イミディエイト減算が検索されることによって変換方式・手順も得られるようにすることができる。例えば、命令フィールドをイミディエイト減算に置き換えるとともに、イミディエイトデータを２の補間を取ったものに置き換えるといった手順を採用することができる。そして、このような手順で得られた命令データを、ハミング距離の比較対象として採用する。また、単にライブラリの検索を行うのではなく、検索が可能であるか否かを判断した後で、可能である場合には検索を行うこととしてもよい。

【０１０７】次に、本実施例の命令列最適化装置の具体的な例について、図２３および図２４を用いて説明する。

【０１０８】図２３は、本実施例の命令列最適化装置が行う最適化処理の手順を説明するためのフローチャートである。

【０１０９】まず、高級言語或いはアセンブリ言語で作製されたソース・プログラムをコンパイルし、さらに他

の最適化処理を施すことにより、中間コード（アセンブリ・コード）を作製する（ステップS2301）。

【0110】次に、各命令について、本実施例を適用する命令であるか否か、すなわちライブラリに置き換え候補が登録されている命令であるか否かを、判断する（ステップS2302）。

【0111】そして、本実施例を適用する命令であると判断された場合は、ライブラリの検索を行って、置換が可能な命令を選出する（ステップS2303）。また、このとき、命令動作を解析して同等の命令を生成することとしてもよい。

【0112】さらに、本実施例を適用する命令およびライブラリで検索された命令について、その前後の命令とのハミング距離を算出する。そして、各算出結果を互いに比較することによって、ハミング距離が最低となるような命令を選択する（ステップS2304）。図24において、（a）は本実施例による最適化を行う前のプログラム例であり、（b）は最適化後のプログラム例である。同図において“addi”命令（イミディエイト加算命令）を“subi”命令（イミディエイト減算命令）に置き換えることにより、その前後の命令との間のハミング距離を26から20に低減させることができた。

【0113】命令の置換が終了すると、続いて、すべてのデータについて最適化が終了したか否かを判定する（ステップS2305）。そして、最適化が終了していないデータが残っている場合には、そのデータについて＊ステップS2303～S2305を実行する。一方、すべてのデータについて最適化が終了している場合には、最適化処理後のプログラムを出力し、最適化処理を終了する。

【0114】本実施例によれば、以上説明したようにして最適化を行った制御プログラムを情報処理装置のプログラムメモリに格納し、この制御プログラムを用いてCPU等の制御を行うことにより、命令バスにおける消費電力を低減させることが可能となる。

【0115】（実施例5）次に、実施例5として、第5の発明の一実施例（請求項9に対応する）について説明する。

【0116】本実施例では、1種類の動作を行うための実現方法が複数ある場合に、作動する機能ブロックが小さく、消費電力が小さくなるなるように、その実現方法に係わる命令を置き換える。すなわち、データ線のばらつき、使用する機能ブロックの消費電力などを考慮し、総合的な消費電力が最小となるように、命令の置き換えを行う。置き換えの方法としては、上述の実施例4の場合と同様、ライブラリを使用することができる。例えば、レジスタ0x0dにデータ“0”を書き込む場合、採用する命令と使用する機能ブロックとの関係は、表1のようになる。これらの命令のうちで、消費電力が最も小さくなるものを選択して、その命令を置き換えることとする。

【0117】

【表1】

命　令	使用する機能ブロック
mov　r0, rd	バイパス回路
add　r0, r0, rd	加算器
mul　r?, r0, rd	乗算器
mul　r0, r?, rd	乗算器
xor　r?, r?, rd	ロジック回路
sll　r0, r?, rd	バレルシフタ
srl　r0, r?, rd	バレルシフタ

次に、本実施例の命令列最適化装置の具体的な例について、図25を用いて説明する。

【0118】図25は、本実施例の命令列最適化装置が行う最適化処理の手順を説明するためのフローチャートである。

【0119】まず、高級言語或いはアセンブリ言語で作製されたソース・プログラムをコンパイルし、さらに他の最適化処理を施すことにより、中間コード（アセンブリ・コード）を作製する（ステップS2501）。

【0120】次に、各命令について、本実施例を適用する命令であるか否か、すなわちライブラリに置き換え候補が登録されている命令であるか否かを、判断する（ス※テップS2502）。

【0121】そして、本実施例を適用する命令であると判断された場合は、ライブラリの検索を行って、置換が可能な命令を選出する（ステップS2503）。また、このとき、命令動作を解析して同等の命令を生成することとしてもよい。

【0122】さらに、本実施例を適用する命令およびライブラリで検索された命令について、消費電力を試算する。そして、各算出結果を互いに比較することによって、消費電力が最低となるような命令を選択する（ステップS2504）。

【0123】命令の置換が終了すると、続いて、すべて

のデータについて最適化が終了したか否かを判定する（ステップS２５０５）。そして、最適化が終了していないデータが残っている場合には、そのデータについてステップS２５０３～S２５０５を実行する。一方、すべてのデータについて最適化が終了している場合には、最適化処理後のプログラムを出力し、最適化処理を終了する。

【０１２４】本実施例によれば、以上説明したようにして最適化を行った制御プログラムを情報処理装置のプログラムメモリに格納し、この制御プログラムを用いてＣＰＵ等の制御を行うことにより、命令バスにおける消費電力を低減させることが可能となる。

【０１２５】

【発明の効果】以上詳細に説明したように、本発明によれば、情報処理装置用制御プログラムの作成段階で消費電力を低減させるための最適化処理を行うことができる、命令列最適化装置を提供することができる。

【図面の簡単な説明】

【図１】実施例１の概念を概略的に示すフローチャートである。

【図２】図１を具体化した例を示すフローチャートである。

【図３】実施例１に係わる命令列最適化装置で最適化されるプログラムの一例を示す図である。

【図４】図３に示したプログラムをコンパイルしたアセンブリ・ソース・プログラムリストを示す図である。

【図５】図３に示したプログラムをコンパイルしたアセンブリ・ソース・プログラムリストを示す図である。

【図６】図３に示したプログラムをコンパイルしたアセンブリ・ソース・プログラムリストを示す図である。

【図７】図３に示したプログラムをコンパイルしたアセンブリ・ソース・プログラムリストを示す図である。

【図８】図３における基本ブロックの依存関係の解析結果を示す有向グラフである。

【図９】図３における基本ブロックの依存関係の解析結果を示す有向グラフである。

【図１０】図３における基本ブロックのビット・パターンを示す図である。

【図１１】実施例１による最適化処理後のビット・パターンを示す図である。

【図１２】実施例１による最適化処理の効果を説明するためのビット・パターンを示す参考図である。

【図１３】実施例１の変形例を説明するためのフローチャートである。

【図１４】（ａ）、（ｂ）ともに、実施例１で最適化されたプログラムを使用する装置の一構成例を示す概念図である。

【図１５】（ａ）、（ｂ）ともに図１４に示した装置で使用するプログラムのビット・パターンを示す図である。

【図１６】（ａ）は実施例１で最適化されたプログラムを使用する装置の一構成例を示す概念図、（ｂ）は（ａ）に示した装置で使用するプログラムのビット・パターンを示す図である。

【図１７】実施例１で最適化されたプログラムを使用する装置の一構成例を示す概念図である。

【図１８】実施例２の命令列最適化装置が行う最適化処理の手順を説明するためのフローチャートである。

【図１９】（ａ）は実施例２の命令列最適化装置が行う最適化処理で使用する中間コードのプログラム例を示す図、（ｂ）は（ａ）のプログラムを最適化した結果を示す図である。

【図２０】図１９（ａ）に示したプログラムの有効範囲テーブルを示す図である。

【図２１】実施例３の命令列最適化装置が行う最適化処理の手順を説明するためのフローチャートである。

【図２２】実施例３の最適化処理を“ａｄｄ”命令について行ったプログラムを示す図である。

【図２３】実施例４の命令列最適化装置が行う最適化処理の手順を説明するためのフローチャートである。

【図２４】（ａ）は実施例４による最適化を行う前のプログラムの一例を示す図、（ｂ）は（ａ）のプログラムを最適化した後のプログラムを示す図である。

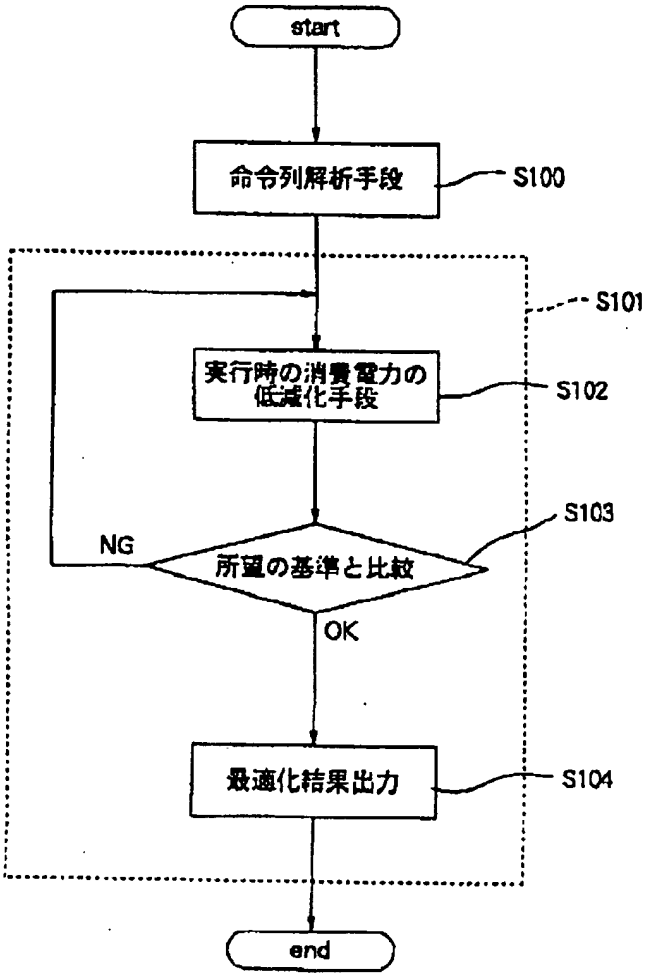
【図２５】実施例５の命令列最適化装置が行う最適化処理の手順を説明するためのフローチャートである。

【図２６】携帯型情報処理装置の制御部の概略構成を示すブロック図である。

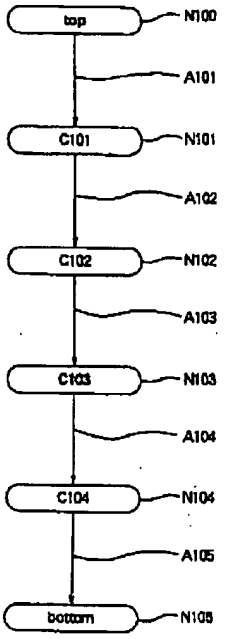
【符号の説明】

２６１０　ＣＰＵ
２６１１　実行ユニット
２６１２　入出力部
２６１３　レジスタ部
２６２０　プログラムメモリ
２６２１　記憶部
２６３１　アドレスバス
２６３２　命令バス

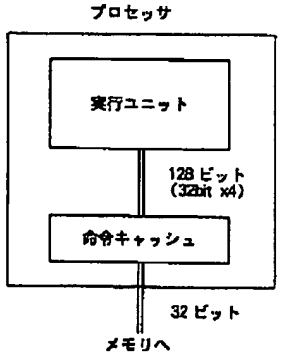
【図1】



【図8】



【図17】



【図10】

C204 : 00000001000000000000000000000000
C301 : 10010000001001111010000110011000
C302 : 11010010000001111011111111111100
C303 : 10010101001010100110000000000010
C304 : 11010000000001000000000000001010
C305 : 10010110001001111010001100101000
C306 : 11011000000001111011111111111100
C307 : 10011011001010110010000000000010
C308 : 11010010000000101100000000001101
C309 : 01000000000000000000000000000000
C310 : 00000001000000000000000000000000
C311 : 11011110000001111011111111111000
C312 : 10100000000001111000000000001000
C313 : 11100000001001111011111111111000

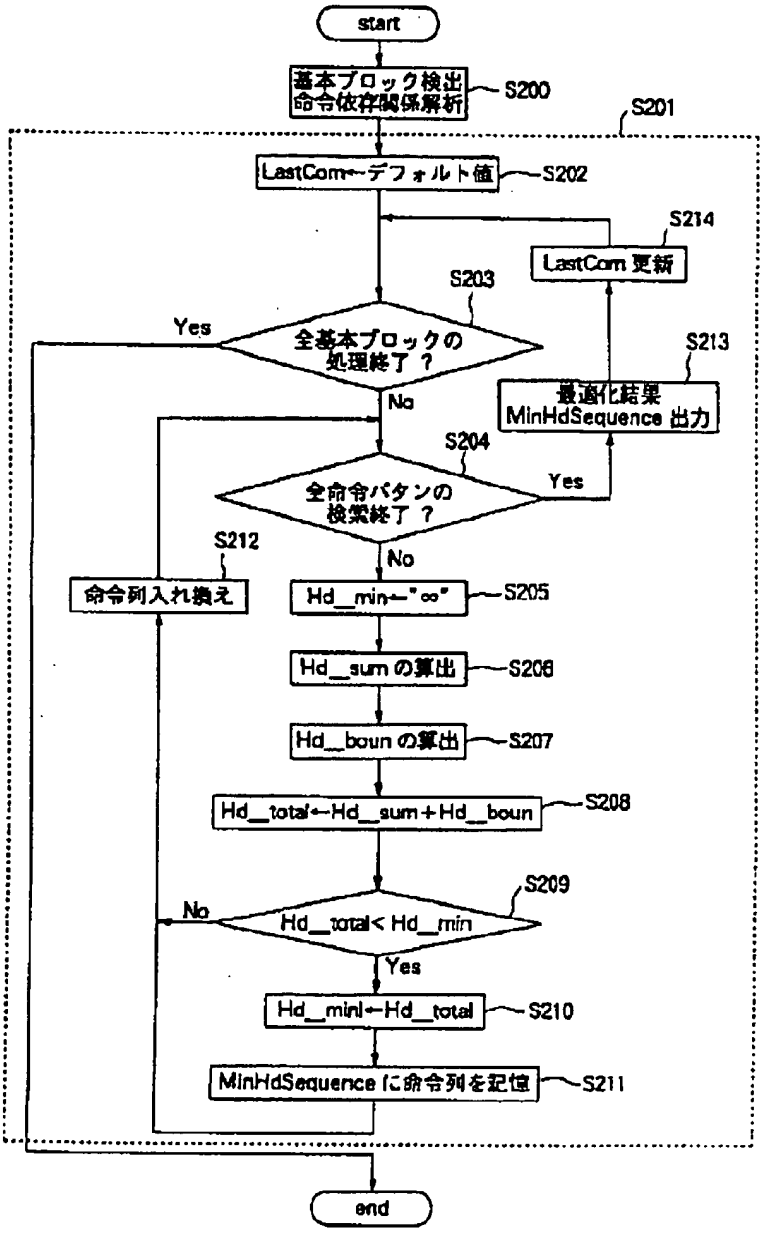
【図11】

C204 : 00000001000000000000000000000000
C301 : 10010000001001111010000110011000
C308 : 11011000000001111011111111111100
C302 : 11010010000001111011111111111100
C305 : 10010110001001111010001100101000
C307 : 10011011001010110010000000000010
C303 : 10010101001010100110000000000010
C304 : 11010000000001000000000000001010
C308 : 11010010000000101100000000001101
C309 : 01000000000000000000000000000000
C310 : 00000001000000000000000000000000
C311 : 11011110000001111011111111111000
C312 : 10100000000001111000000000001000
C313 : 11100000001001111011111111111000

【図20】

レジスタ番号	開始アドレス	終了アドレス
0x01	0011	0101
0x02	0100	0110
0x03	0110	0111
0x04	0000	0110
0x05	0001	1011
0x06	0010	1100
0x1c	0101	1000
0x01	1011	1101
0x02	1100	1101
0x1c	1101	1110

【図2】



【図22】

div r13,r16,r11	010111 10001 10011 000 11110101 10110
add r11,r5,r17	111111 10111 10001 010 00000000 00101
div r17,r14,r14	010111 10101 10011 000 11110101 10111
add r11,r5,r17	000000 10111 10001 010 00000000 00101
mul r17,r14,r14	010010 10101 10011 000 11110101 10111

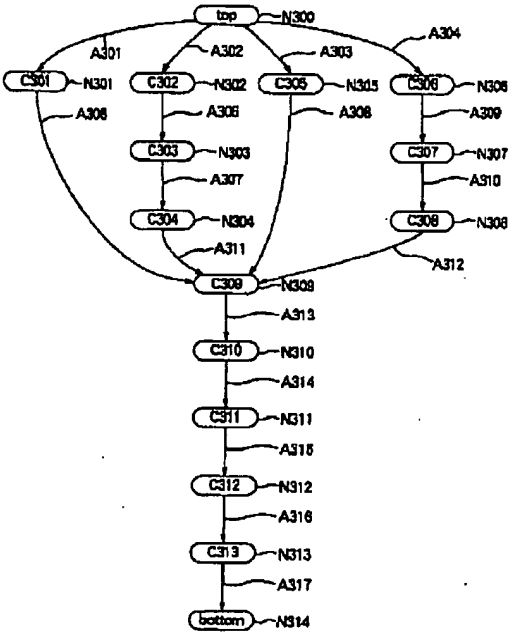
【図3】

```
/* Sample Program */

#include      <stdio.h>
#define       MAX   100

main()
{
      int   i, q, z[MAX], x[max] ;
      for (i=0; i < MAX; i++) q = q + z[i] * x[i] ;
      printf("Q = Q + Z[i] x X[i] : %d¥n", q);
}
```

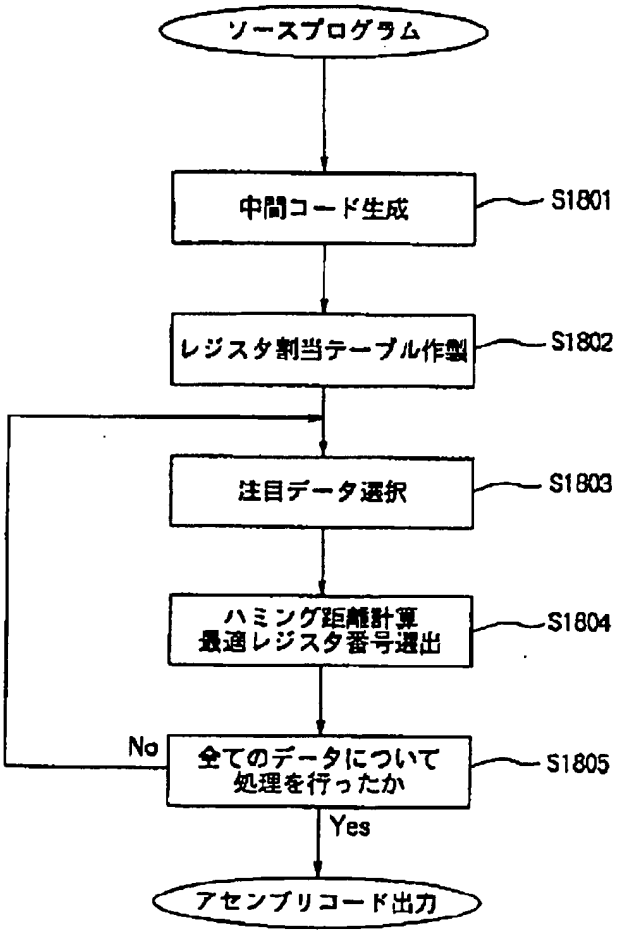
【図12】

```
C204 : 00000001000000000000000000000000
C302 : 11010010000001111011111111111100
C303 : 10010101001010100110000000000010
C306 : 11011000000001111011111111111100
C307 : 10011011001010110010000000000010
C301 : 10010000001001111010000110011000
C308 : 11010010000000101100000000001101
C305 : 10010110001001111010001100101000
C304 : 11010000000000100000000000001010
C309 : 01000000000000000000000000000000
C310 : 00000001000000000000000000000000
C311 : 11011110000001111011111111111000
C312 : 10100000000000111100000000001000
C313 : 11100000001001111011111111111000
```

【図9】



【図18】



【図4】

```
*********************************************
LineNo  Address     Object Code     Assembley Source
*********************************************
   1                                .seg    "data"
   2                                .seg    "text"
   3                                .proc 04
   4                                .global __main
   5                            __main:
   6                            LL0:
   7                                ! # PROLOGUE # 0
   8  00000000  033fffff            sethi   %hi(LF25),%g1  ---------- C101 ┐
   9  00000004  82006078            add     %g1,%lo(LF25),%g1  ------- C102 ├ B1
  10  00000008  9de38001            save    %sp,%g1,%sp  ------------- C103 ┘
  11                                ! # PROLOGUE # 1
  12  0000000c  c027bffc            st      %g0,[%fp+-0x4]  ---------- C104
  13                            L29:
  14  00000010  d007bffc            ld      [%fp+-0x4],%o0  ---------- C201 ┐
  15  00000014  80a22064            cmp     %o0,0x64  ---------------- C202 ├ B2
  16  00000018  16800014            dge     L28  --------------------- C203 ┘
  17  0000001c  01000000            nop  ---------------------------- C204
```

【図5】

```
18  00000020  9027a198      sub   %fp, 0x198, %o0 ---------- C301   } B3
19  00000024  d207bffc      ld    [%fp+-0x4], %o1 ---------- C302
20  00000028  952a6002      sll   %o1, 0x2, %o2 ---------- C303
21  0000002c  d0020000      ld    [%o0+%o2], %o0 ---------- C304
22  00000030  9627a328      sub   %fp, 0x328, %o3 ---------- C305
23  00000034  d807bffc      ld    [%fp+-0x4], %o4 ---------- C306
24  00000038  9b2b2002      sll   %o4, 0x2, %o5 ---------- C307
25  0000003c  d202c00d      ld    [%o3+%o5], %o1 ---------- C308
26  00000040  40000000      call  .mul, 2 ---------- C309
27  00000044  01000000      nop                     C310
28  00000048  de07bff8      ld    [%fp+-0x8], %o7 ---------- C311
29  0000004c  a003c008      add   %o7, %o0,%l0 ---------- C312
30  00000050  e027bff8      st    %l0, [%fp+-0x8] ---------- C313
31                      L27 :
32  00000054  e207bffc      ld    [%fp+-0x4], %l1           } B4
```

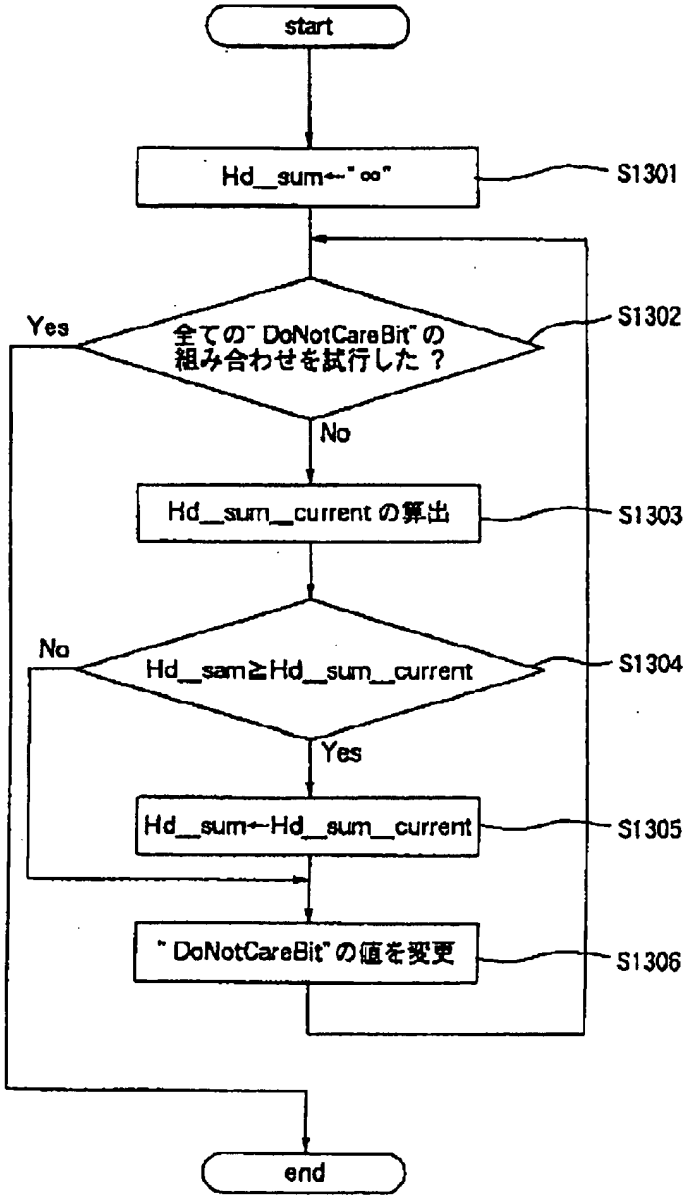
【図6】

```
                                    add   %l1, 0x1, %l1       } B4
                                    st    %l1, [%fp+-0x4]
                                    b     L28
                                    nop
                            L28 :
                                    .seg  ".data1"
                            L31 :
                                    .ascii "Q = Q + Z[i] x X[i] :
                                    .seg  ".text"             } B5
                                    set   L31, %o0
                                    ld    [%fp+-0x8], %o1
                                    call  _printf, 2
                                    nop
                                    mov   0, %o0
33  00000058  a2046001      
34  0000005c  e227bffc      
35  00000060  10bfffec      
36  00000064  01000000      
37
38
39
40
%d*012*0"
41
42  00000068  11000000      
42  0000006c  90122000      
43  00000070  d207bff8      
44  00000074  40000000      
45  00000078  01000000      
46  0000007c  90102000      
```

【図7】

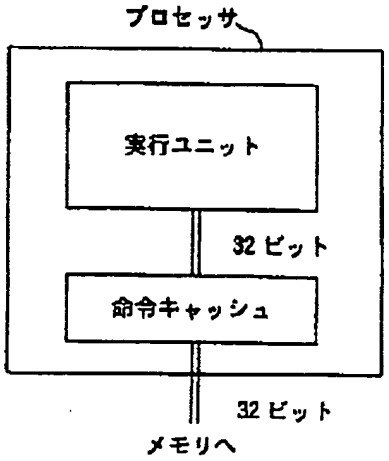
```
47                                  LE25 :   mov      %o0, %i0    ┐
48    00000080     b0100008                  ret                  ├ B6
49    00000084     81c7e008                  restore              ┘
50    00000088     81e80000                 LF25 = -904
51                                           LP25 = 96
52                                           LT25 = 96
53                                           .seg       "data"
54
55
56
*************** LABEL  LIST *****************
1                              LL0          0
6                            __main         0
13                              L29        10
31                              L27        54
37                              L28        68
39                              L31         0
47                             LE25        80
51                             LF25   ffffc78
52                             LP25        60
```

【図13】

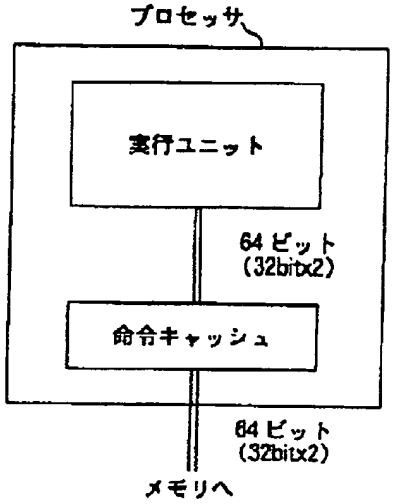

start
Hd_sum←"∞"
S1301
全ての"DoNotCareBit"の組み合わせを試行した？
S1302
Yes
No
Hd_sum_current の算出
S1303
Hd_sam≧Hd_sum_current
S1304
No
Yes
Hd_sum←Hd_sum_current
S1305
"DoNotCareBit"の値を変更
S1306
end

【図14】

(a)



(b)



【図24】

(a)

```
mul r13,r16,r11      010010 10001 10011 000 11110101 10110
addi r11,5,r17       010000 10111 10001 010 0000000000101
mul r17,r14,r14      010010 10101 10011 000 11110101 10111
```

(b)

```
mul r13,r16,r11      010010 10001 10011 000 11110101 10110
subi r11,5,r17       010001 10111 10001 010 1111111111011
mul r17,r14,r14      010010 10101 10011 000 11110101 10111
```

【図15】

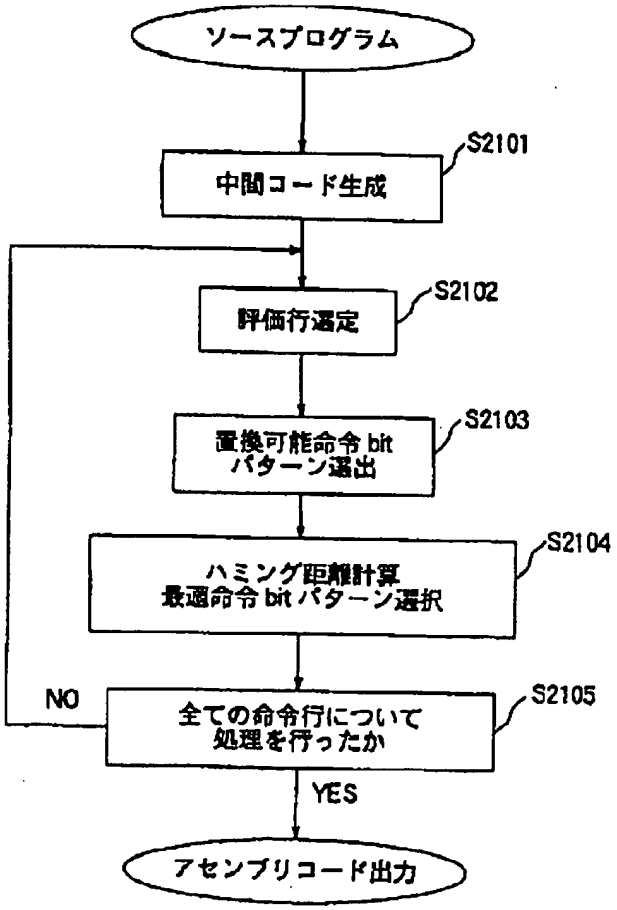
(a)

C301 : 1001000000100111101000011001 1000
C302 : 1101001000000111101111111111 1100
C303 : 1001010100101010011000000000010
C304 : 1101000000000010000000000001010
C305 : 1001011000100111101000110010 1000
.....
C310 : 0000000100000000000000000000000
C311 : 1101111000000111101111111111 1000
C312 : 1010000000000011110000000001000
C313 : 1110000000100111101111111111 1000

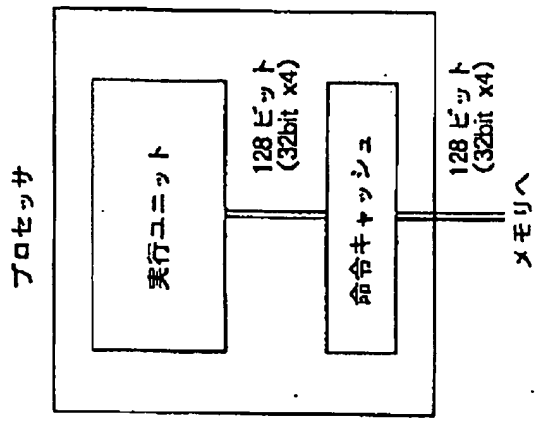
(b)

C301, C302 : 1001000000100111101000011001 1000 1101001000000111101111111111 1100
C303, C304 : 1001010100101010011000000000010 1101000000000010000000000001010
.....
C310, C311 : 0000000100000000000000000000000 1101111000000111101111111111 1000
C312, C313 : 1010000000000011110000000001000 1110000000100111101111111111 1000

【図21】



【図16】



(a)

C301, C302, C303, C304 : 10010..11000 11010..11100 10010.00010 11010.01010

C309, C310, C311, C312 : 01000..00000 00000..00000 11011..11000 10100..01000

(b)

【図19】

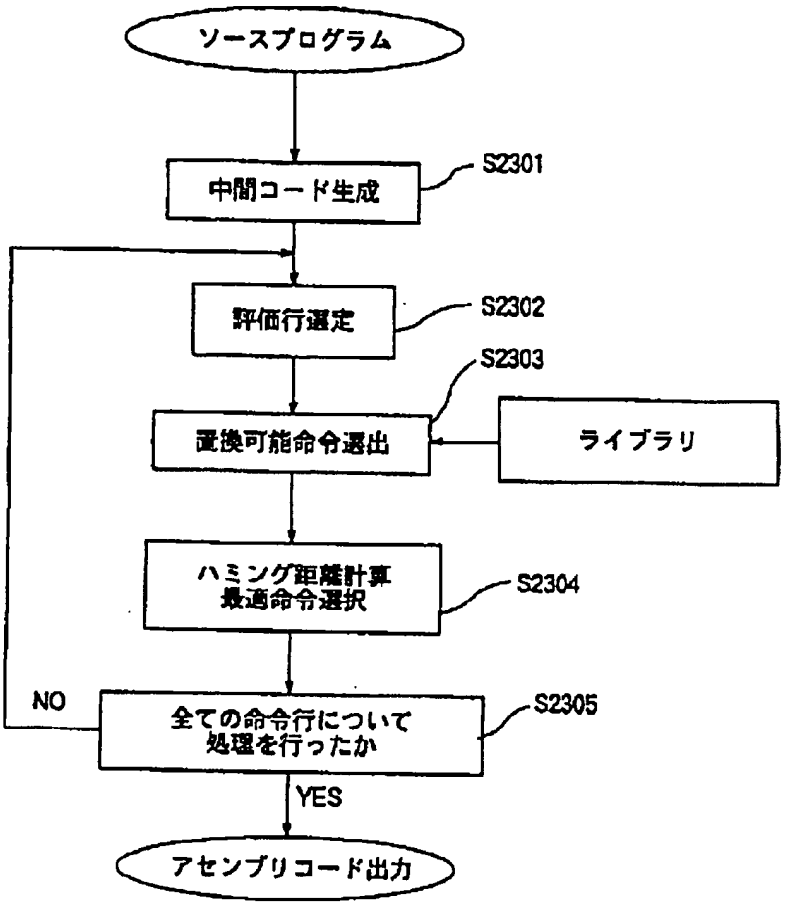
(a)

```
LABEL :
0000    set  0x00101  r4     1011010000 00100 0000000001000 00001
0001    set  0x00215  r5     1011010000 00101 0000000010000 10101
0010    set  0x00112  r6     1011010000 00101 0000000001000 10010
0011    ld [r5+0x4] r1       1100000001 00001 00101 0000000 00100
0100    ld [r6+0x4] r2       1100000001 00010 00110 0000000 00100
0101    mul r1 r2 r1c        1100101011 11100 00001 0000000 00010
0110    add r2 r4 r3         1100101011 00011 00010 0000000 00100
0111    sub r1c r3 r1c       1100011011 11100 00011 0000000 11100
1000    cmp r1c 0x4c         1011010000 11100 00000 0000010 01100
1001    bne LABEL            0100000010 00000 11111 1111111 11100
1010    nop                  0000000000 00000 00000 0000000 00000
1011    ld [r5+0x20] r1      1100000001 00001 00101 0000001 00000
1100    ld [r6+0x20] r2      1100000001 00010 00110 0000001 00000
1101    mul r1 r2 r1c        1100101011 11100 00001 0000000 00010
1110    st [r6+0x28] r1c     1100100111 11100 00100 0000001 01000
```

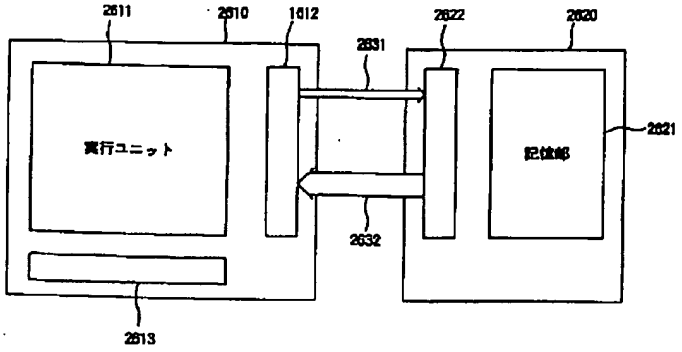
(b)

```
LABEL :
0000    set  0x00101  r4     1011010000 00100 0000000001000 00001
0001    set  0x00215  r5     1011010000 00101 0000000010000 10101
0010    set  0x00112  r6     1011010000 00101 0000000001000 10010
0011    ld [r5+0x4] r1       1100000001 00001 00101 0000000 00100
0100    ld [r6+0x4] r2       1100000001 00010 00110 0000000 00100
0101    mul r1 r2 r0         1100101011 00000 00001 0000000 00010
0110    add r2 r4 r3         1100101011 00011 00010 0000000 00100
0111    sub r0  r3 r0        1100011011 00000 00011 0000000 00000
1000    cmp r0 0x4c          1011010000 00000 00000 0000010 01100
1001    bne LABEL            0100000010 00000 11111 1111111 11100
1010    nop                  0000000000 00000 00000 0000000 00000
1011    ld [r5+0x20] r1      1100000001 00001 00101 0000001 00000
1100    ld [r6+0x20] r2      1100000001 00010 00110 0000001 00000
1101    mul r1 r2 r1c        1100101011 11100 00001 0000000 00010
                        (有効範囲外なので変更せず)
1110    st [r6+0x28] r1c     1100100111 11100 00100 0000001 01000
```

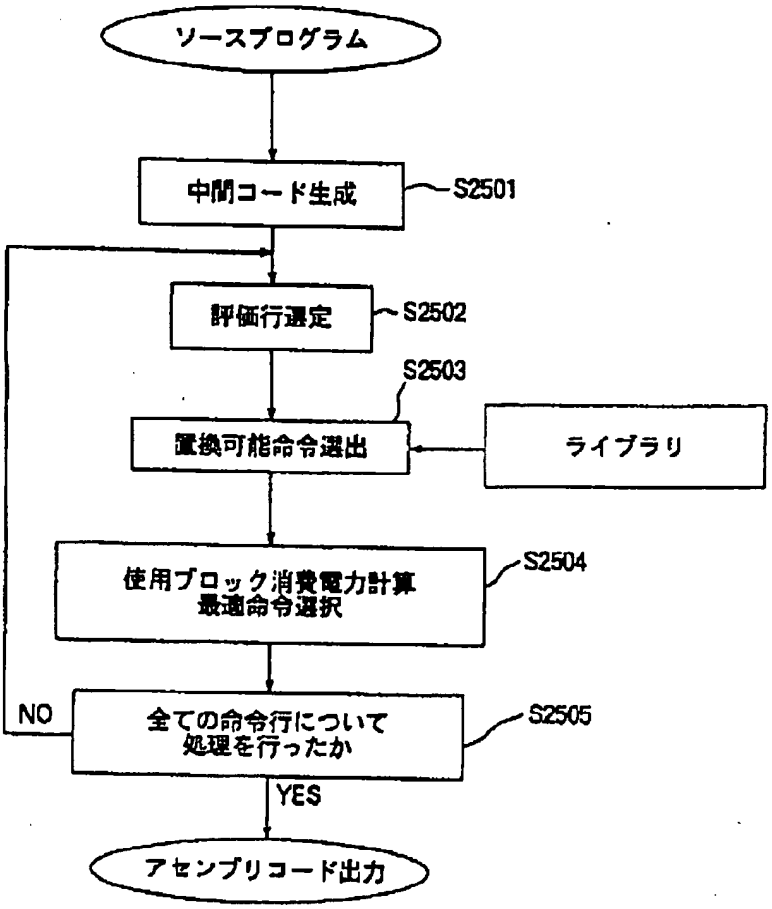
【図23】


ソースプログラム
中間コード生成
S2301
評価行選定
S2302
S2303
置換可能命令選出
ライブラリ
ハミング距離計算
最適命令選択
S2304
NO
全ての命令行について
処理を行ったか
S2305
YES
アセンブリコード出力

【図26】


2611
2610
1612
2622
2620
2631
実行ユニット
2621
2632
2613

【図25】


ソースプログラム
中間コード生成
S2501
評価行選定
S2502
S2503
置換可能命令選出
ライブラリ
使用ブロック消費電力計算
最適命令選択
S2504
NO
全ての命令行について
処理を行ったか
S2505
YES
アセンブリコード出力